**REPORTE PROGRAMACIÓN AVANZADA**
**PROYECTO VIDEOJUEGO**
**JUEGO “AUTITOS”**

Profesor:
Claudio Cubillos

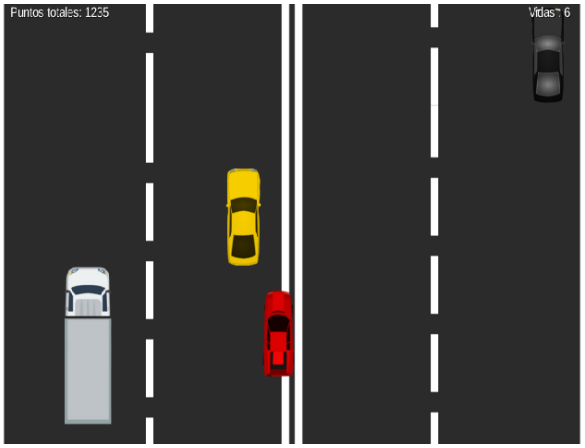
Victor Huerta
Davor Serey
Ignacio Silva

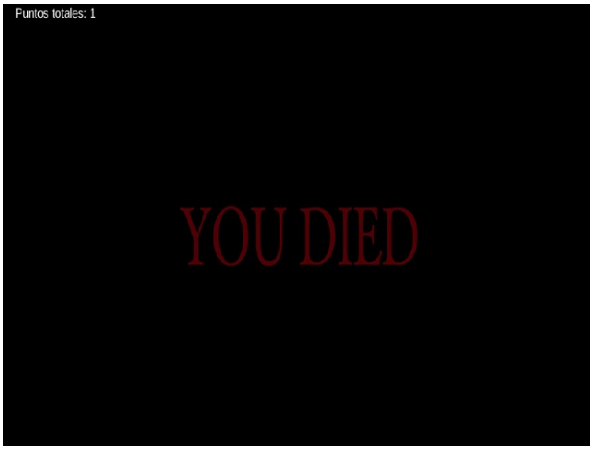
**Valparaíso**
30 de Octubre, 2022

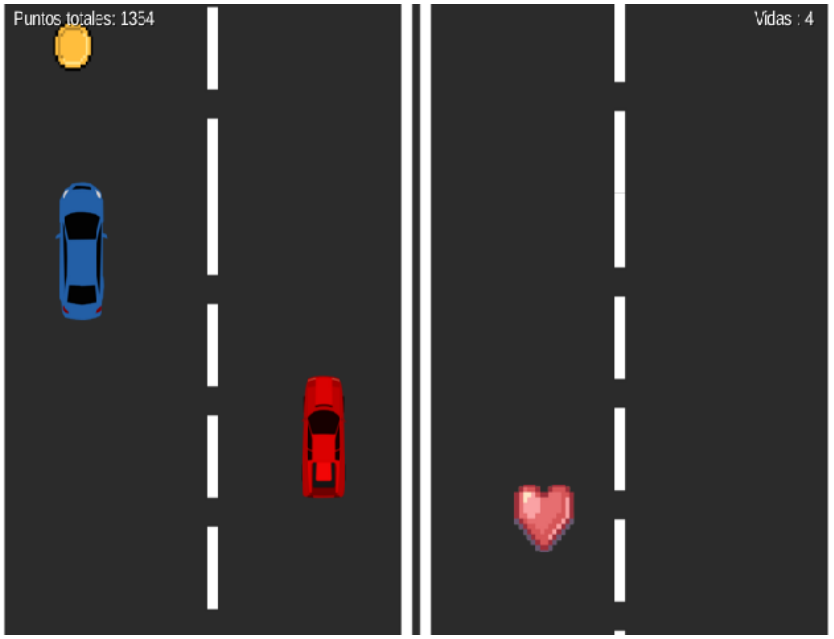
**1.1**

## Tutorial Autitos

Al momento de iniciar el juego, el jugador va a tener en frente un auto rojo que es posible moverlo en todas las direcciones  a través de las flechas del teclado del usuario. El jugador tendrá su puntaje que aumentará con el tiempo y su vida, inicialmente tomará el valor de 3.
Como imagen de fondo va a haber una carretera. Además aparecerán obstáculos tales como autos amarillos, azules y negros, y también camiones. Además aparecerán corazones y monedas.
Los autos y camiones si chocas con ellos perderás vidas, si estas llegan a 0, pierdes el juego. Los corazones te otorgan una vida más si los agarras, y por último son las monedas que una vez vas agarrando las monedas te van otorgando 25 puntos adicionales.
Si pierdes el juego, se mostrará una pantalla de Game Over.



**1.2**

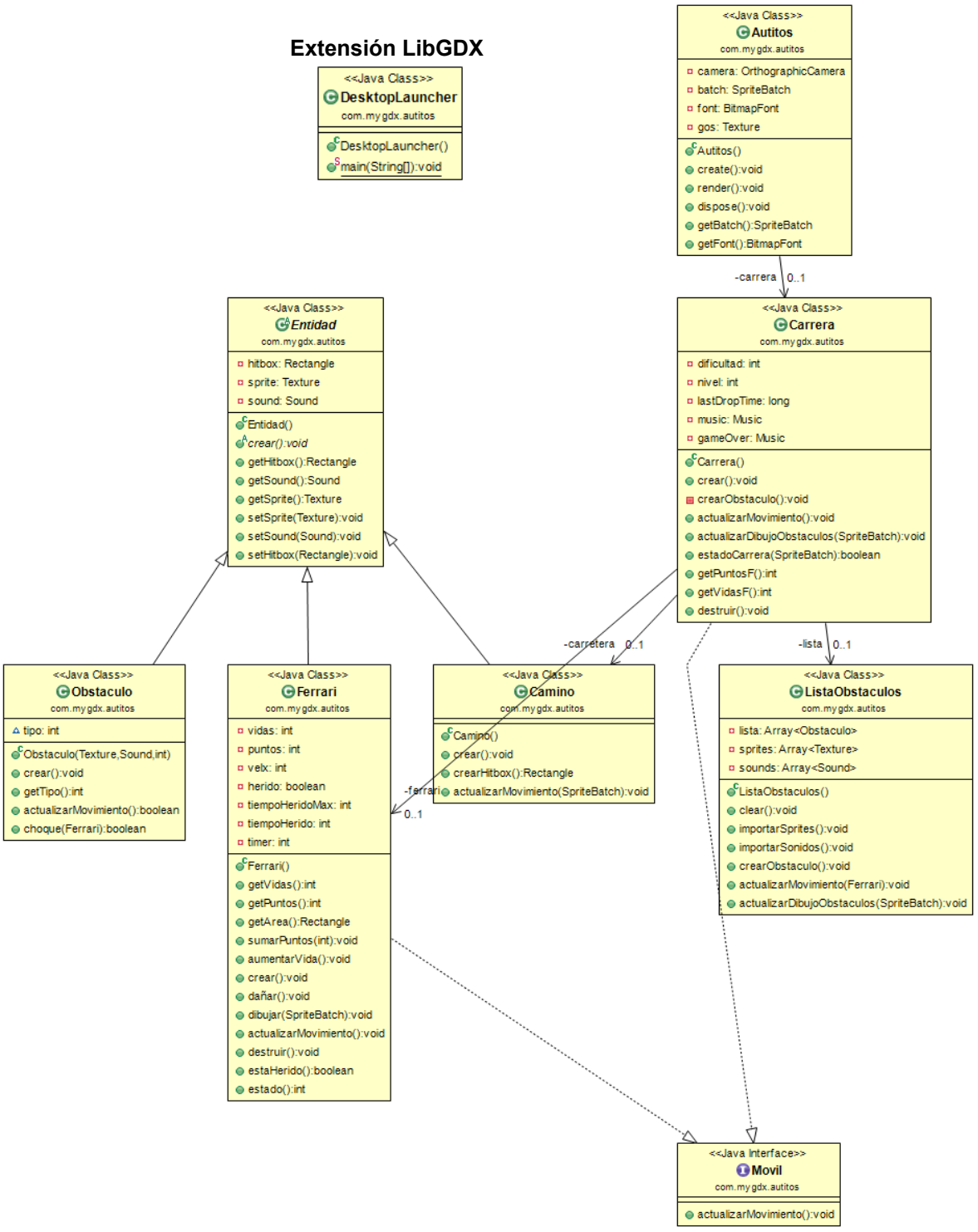
## Juego base y modificaciones

El juego inicial que ocupamos como base es el juego GameLluvia. Éste consistía en un tarro recogiendo gotas de agua que iban apareciendo y así sumando puntos, y a la vez esquivando las gotas rojas que te van descontando puntos y vida. La finalidad del juego es llegar al mayor puntaje posible. Para este proyecto, se eligió el juego base explicado anteriormente y se hicieron los siguientes cambios:

1. Se creó una clase abstracta Entidad para facilitar dibujar y mantener en movimiento a los objetos en pantalla.
2. Se cambió el tarro que mueve el jugador por un auto rojo (Clase Ferrari), agregando mayor movilidad, pudiendo desplazarse en las cuatro direcciones. Esta clase se extiende de Entidad.
3. Se hicieron cambios en el sistema de puntaje. Antes aumentaba sólo al atrapar gotas azules. Ahora con el tiempo se irá aumentando gradualmente el puntaje del usuario, representando la distancia que recorre el auto rojo.
4. En el juego base existen gotas de distinto color. Ahora se creó la clase Obstáculo que sirve para crear y manipular a los distintos objetos que aparecen aleatoriamente durante el juego. Esta clase se extiende de Entidad, que permite almacenar la hitbox, el sprite y el sonido de cada objeto y utilizarlo cuando se necesite con mayor facilidad.
5. En el punto anterior se presentó la clase Obstaculo, que esta se constituye de los siguientes objetos:
    a. Autos amarillos, rojos y camiones, los cuáles si el usuario colisiona, le resta vida.
    b. Monedas, las cuáles dan 25 puntos adicionales al jugador.
    c. Corazones que aumentan en uno la vida del usuario.
6. Se crea una clase ListaObstaculos donde se importan todos los sprites y sonidos de los objetos, y se almacenan en Arrays, además se crea un Array de Obstáculos para contener a los objetos que está procesando el juego. Cuando el usuario colisiona con un obstáculo o éste sale de la pantalla, se elimina dicho objeto, tal y como lo hace el juego base. Esta clase sirve para encapsular los objetos del juego.
7. En el juego base, el fondo es de color azul marino. Se cambió a una carretera que se encuentra en movimiento, acompañado de una canción.. Al perder el juego, se muestra una pantalla de game over que dice “You Died” y también cambia la canción.
8. El juego presenta un cambio de dificultad durante el proceso. Cada 500 puntos se aumentará un poco la dificultad, aumentando la cantidad de objetos que aparecen. Se irá aumentando la dificultad hasta un límite de 5000 puntos.

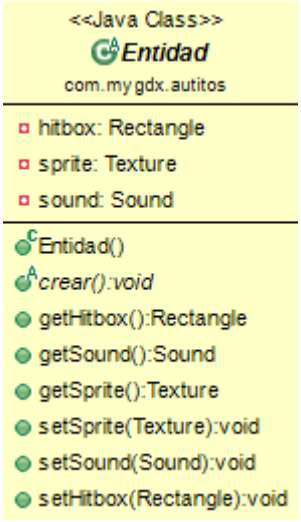
## 1.3

# Diseño de diagrama UML



**1.4**

## Clase Abstracta




Esta clase Entidad se utiliza para poder crear cada objeto que se utiliza en este juego (autos, camión, moneda, corazón). Cada uno contiene su textura (imagen), sonido y su hitbox (tamaño objeto). Esta clase abstracta tiene getters y setters para que sus subclases puedan tener acceso a los atributos que esta entrega. El método sin implementación que tiene Entidad es **crear()**, donde las clases hijas la sobreescriben e implementan a su modo.
Las clases que se extienden de Entidad son Ferrari y Obstaculo, donde se facilita el almacenamiento e implementación de los sprites, hitbox y sonido de cada objeto en pantalla.
Al momento de dibujar en pantalla todos los obstáculos, se utiliza esta clase abstracta, ya que permite generalizar la obtención de los datos a utilizar.

**1.5**

## Interfaz

Durante la ejecución del juego, algunos objetos necesitan mantenerse en movimiento, dibujándolos para que la pantalla muestre el estado actual del juego para que el usuario pueda visualizar qué está ocurriendo. Se creó una interfaz con un método sin implementación llamado **actualizarMovimiento( )**. Las clases Ferrari y Carrera, donde sobreescriben el método de la interfaz e implementan su propio método. Esto sirve como guía para estas dos clases.

**1.6**

## Encapsulamiento principios OO

Como se puede observar cada clase de este proyecto se encarga de hacer por sí misma lo que se relacione con su clase que quiere decir esto. Un ejemplo de esto, es que en la clase carrera, al momento de que el auto del usuario choca con un elemento dañino, se llama al método dañar que lo que hace es quitarle una vida a este, en vez de que en la misma clase Carrera hacer que se reste una vida, entonces con esto se puede decir que se tiene el código encapsulado. También, la clase Carrera, donde se hacen los procesos para que el juego funcione, mantiene una encapsulación de detalles de diseño, ya que esta mueve todo lo que ocurre durante el juego, y si se hacen cambios, no perjudica el juego por completo.

**1.7**

## Commits (Github)

Durante el progreso del proyecto, se utilizó GitHub para mantener un control de versiones. Se hicieron un total de 11 commits.

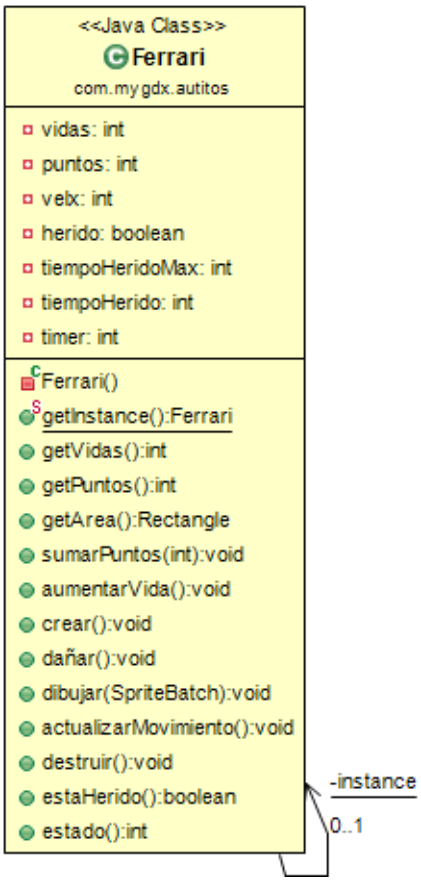
**Fechas:**

1. Octubre 25.2022 (“Initial commit”, “Creacion Juego”)
2. Octubre 27.2022 (“Add files via upload”, “Update”)
3. Octubre 28.2022 (“Abstraccion hecha codigo”, “Ajueste diseño”, “Update”)
4. Octubre 29.2022 (“Mas tipos de obstaculos”, “Implementacion funciones”)
5. Octubre 30.2022 (“Musica GameOver”, “Creacion Interfaz”)

**2.1**

## Singleton

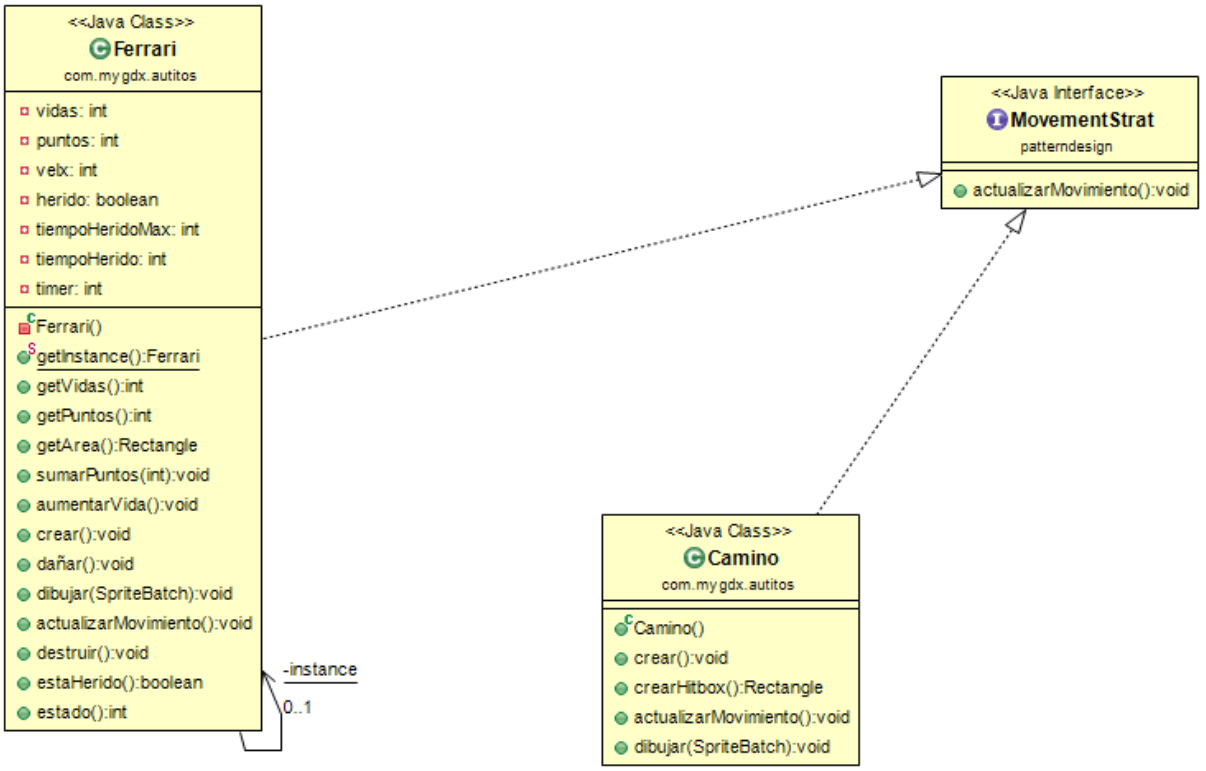
El objeto principal del videojuego es el automóvil que controla el usuario, el “Ferrari” rojo, y debe existir sólo uno dentro de todo el juego, es por esto que se implementó Singleton a la clase Ferrari, para asegurarse de que el auto rojo del usuario sea único y sólo exista una instancia de esta clase. Dentro de la clase Ferrari se creó una instancia estática de sí misma, y se cambió el constructor a privado, para que ninguna otra clase pueda instanciar un Ferrari. Además, se creó un método estático sincronizado “getInstance”, lo que hace es, que si la instancia es nula, entonces se llama al constructor, se crea el objeto y se retorna, pero si este no es nulo, retorna la instancia que ya se generó con anterioridad.



**2.3**

## Strategy

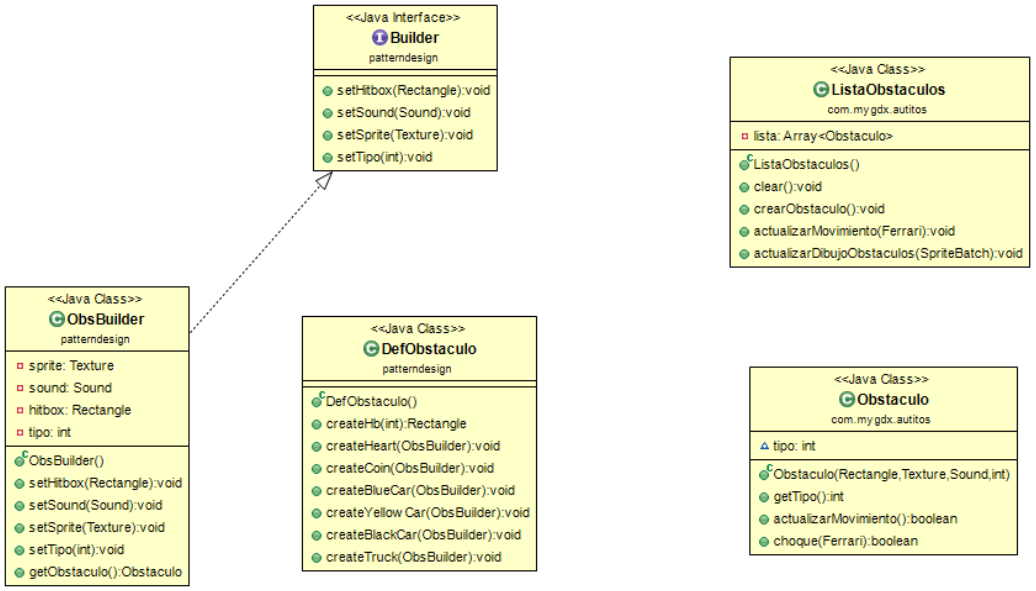
En el código del juego, la mayoría de los objetos se mueven, y lo hacen de distintas formas, por lo que para generalizar y facilitar la comprensión del código, se implementó el patrón de diseño “Strategy”, de tal forma que para la mayoría de objetos que se muevan, se implementa la interfaz “MovementStrat”, que permite que las clases como Ferrari o Carretera actualicen su posición y moverse, de tal forma que durante el desarrollo del juego, el usuario pueda observar cómo se mueven los elementos en pantalla.



**2.4**

## Builder

Los obstáculos son los objetos que van apareciendo desde arriba, y pueden ser beneficiosos o perjudiciales para el usuario, ya que existen varios obstáculos, donde cada uno tiene distintas características, sonidos, texturas y utilidades. Debido a esto, se utilizó el patrón de diseño “Builder”, para mantener limpio y generalizado el código, y así mantener más controlado la creación de obstáculos. En específico, se creó una interfaz “Builder”, donde se tienen los métodos sin implementación, también se creó la clase “ObsBuilder” que implementa a “Builder”, donde se sobreescriben e implementan los métodos para el correcto funcionamiento, y finalmente se creó la clase “DefObstaculo”, lo que hace esta clase es definir las características de los distintos obstáculos, y generarlos para el videojuego, utilizando el builder de los obstáculos.




**2.5**

## Commits (Github)

Además de los commits hechos para la entrega de avance de proyecto del juego, se hicieron otros 4 commits durante el desarrollo de la entrega final del videojuego.

**Fechas:**

1. Diciembre 1.2022 (“Implementacion Builder y Singleton”)
2. Diciembre 2.2022(“Update README”, “Implementacion Strategy”, “Ultima actualizacion”)